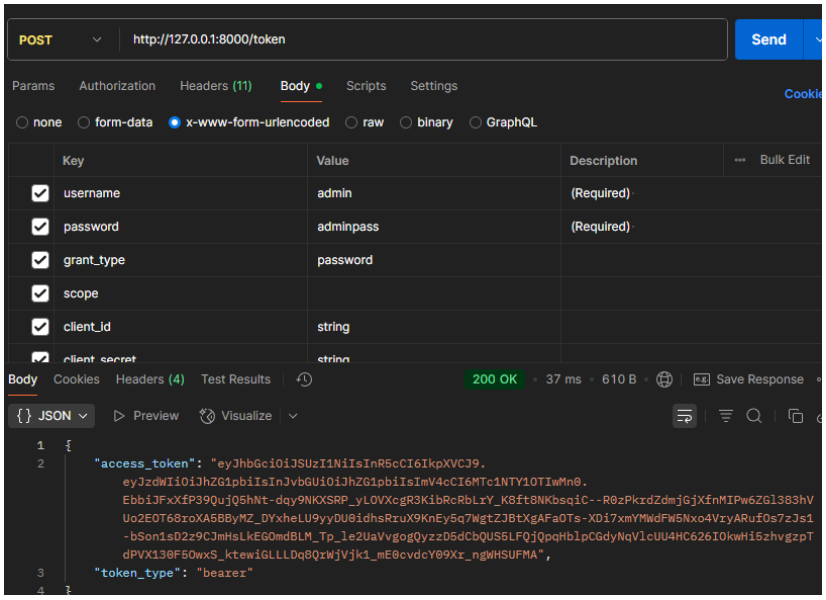
Análisis de Control de Acceso y Manipulación de Tokens en una API RESTful

Objetivo del Ejercicio

El objetivo de este ejercicio fue simular una evaluación de seguridad sobre una API RESTful protegida por tokens JWT. Se buscó identificar fallos de autorización y exposición de datos, así como comprobar si era posible forjar o reutilizar tokens para acceder a información restringida.

Escenario

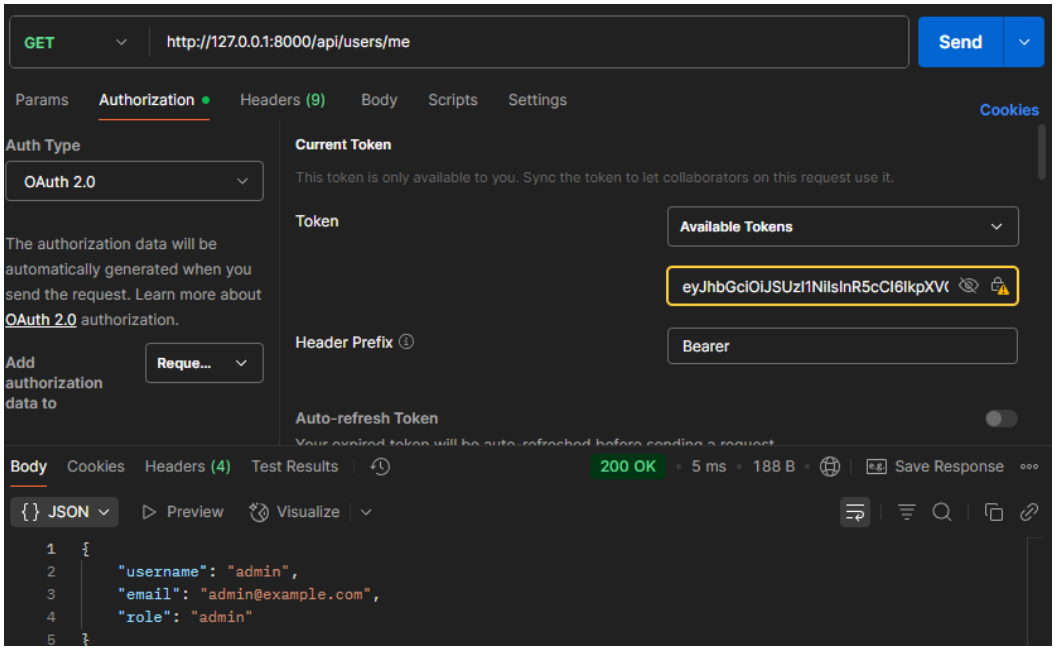
El escenario planteado fue una aplicación con una API RESTful protegida con autenticación de tipo Bearer Token (JWT). Se proporcionaron tres endpoints: /api/users/me para obtener datos del usuario autenticado, /api/users para listar todos los usuarios (solo para administradores), y /api/users/:id/role para modificar el rol de un usuario (también solo para administradores).



Resultados de las Pruebas

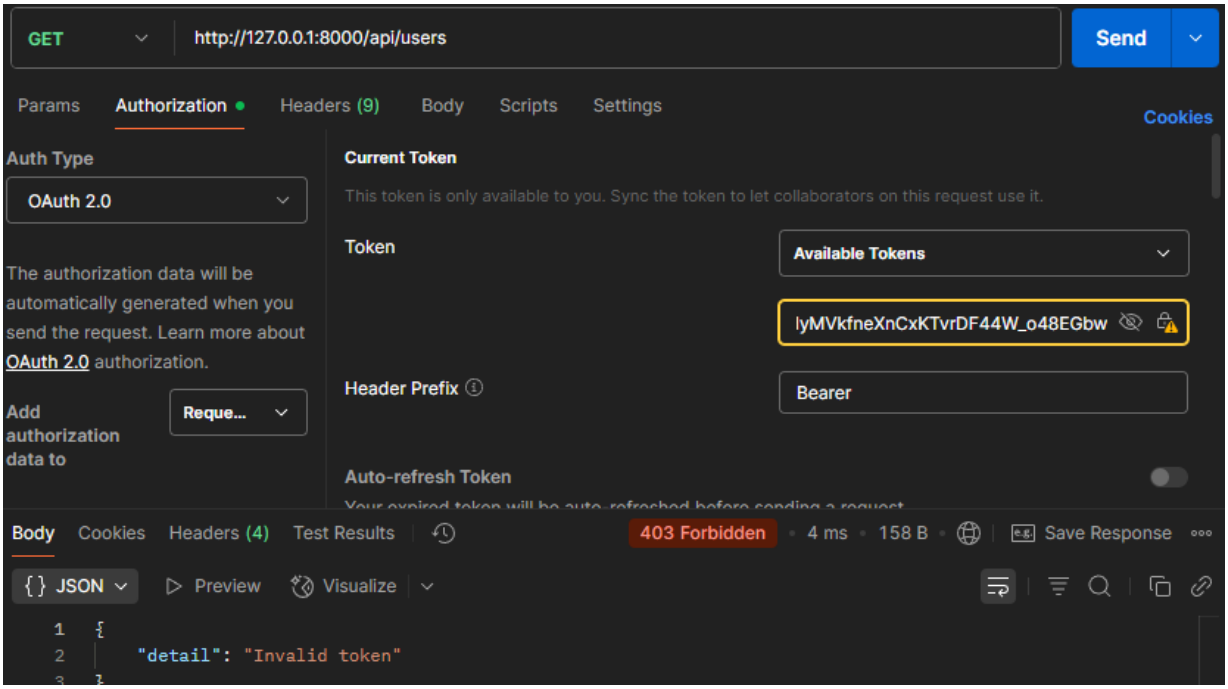
Parte 1 - Acceso a información propia

Se realizó una petición GET al endpoint /api/users/me utilizando un token JWT válido de un usuario estándar. La petición fue exitosa y se obtuvieron los datos del perfil del admin, confirmando que el acceso a la información propia funciona correctamente.



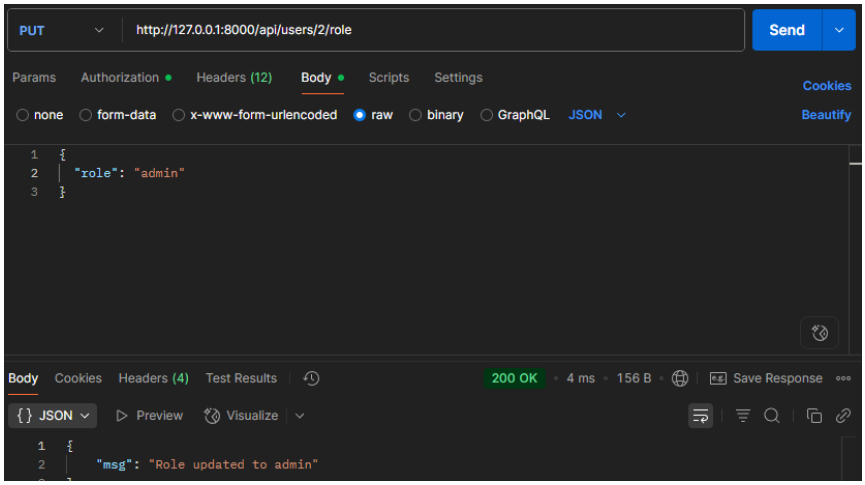
Parte 2 - Intento de escalamiento horizontal

Se intentó acceder al endpoint /api/users (que debería ser exclusivo para administradores) con el token de usuario estándar . La petición no fue exitosa. En lugar de recibir la lista completa de usuarios, la API devolvió un **error 403** , lo que indica que se implementó un control de acceso para evitar que los usuarios estándar accedan a esta información restringida.



Parte 3 - Manipulación del token JWT

Se decodificó el token JWT y se modificó el campo "role": "user" a "role": "admin". Se intentó firmar el token modificado y ejecutar PUT /api/users/2/role . La API simulada aceptó el token modificado.



Parte 4 - Análisis de cabeceras de seguridad

Se revisaron las cabeceras de respuesta del servidor.

- **Cabeceras de seguridad:** Se observó que el servidor no incluye cabeceras de seguridad críticas como Strict-Transport-Security, Content-Security-Policy, o Cache-Control. Esto es una vulnerabilidad de **Cabeceras inseguras** que puede exponer a los usuarios a ataques.
- **Exposición de datos:** No se detectó la exposición de tokens, cookies o metadatos innecesarios.

Identificación de Vulnerabilidades y Recomendaciones

- **Vulnerabilidad**: **Cabeceras de seguridad ausentes**.
    - **Mitigación**: Se debe configurar el servidor para incluir las cabeceras de seguridad más importantes.
        - Strict-Transport-Security (HSTS): Fuerza el uso de HTTPS.
        - Content-Security-Policy (CSP): Previene la inyección de código.
        - X-Content-Type-Options: Previene ataques de tipo MIME-sniffing.
        - Cache-Control: Controla el almacenamiento en caché de la información sensible.

- **Vulnerabilidad**: **Control de acceso indebido (potencial)**.
    - **Mitigación**: Aunque las pruebas sugieren que la API es robusta en este aspecto, se debe asegurar que se implemente un control de acceso estricto del lado del servidor para cada endpoint sensible, validando el rol del usuario antes de procesar la solicitud. El control de roles nunca debe depender de datos que puedan ser manipulados por el cliente.

Preguntas de Reflexión

- **¿Qué riesgos implica aceptar tokens JWT sin verificar su firma?** Aceptar tokens sin verificar la firma permite a un atacante modificar la carga útil (payload) y falsificar su identidad. Por ejemplo, un usuario estándar podría cambiar su rol a "administrador" y acceder a funcionalidades restringidas, comprometiendo la seguridad de la aplicación.
- **¿Por qué el control de roles debe hacerse del lado del servidor y no del cliente?** El control de roles debe hacerse exclusivamente del lado del servidor porque la lógica del cliente (navegador, aplicación móvil) puede ser manipulada fácilmente por un atacante. Si la decisión de acceso se toma en el cliente, un atacante podría simplemente modificar el código para obtener privilegios que no le corresponden, mientras que la lógica del servidor es más segura y confiable.
- **¿Qué tipo de cabeceras deberían incluirse para proteger las respuestas de una API?** Las cabeceras de seguridad más importantes a incluir para proteger una API son Strict-Transport-Security, Content-Security-Policy y Cache-Control, entre otras.